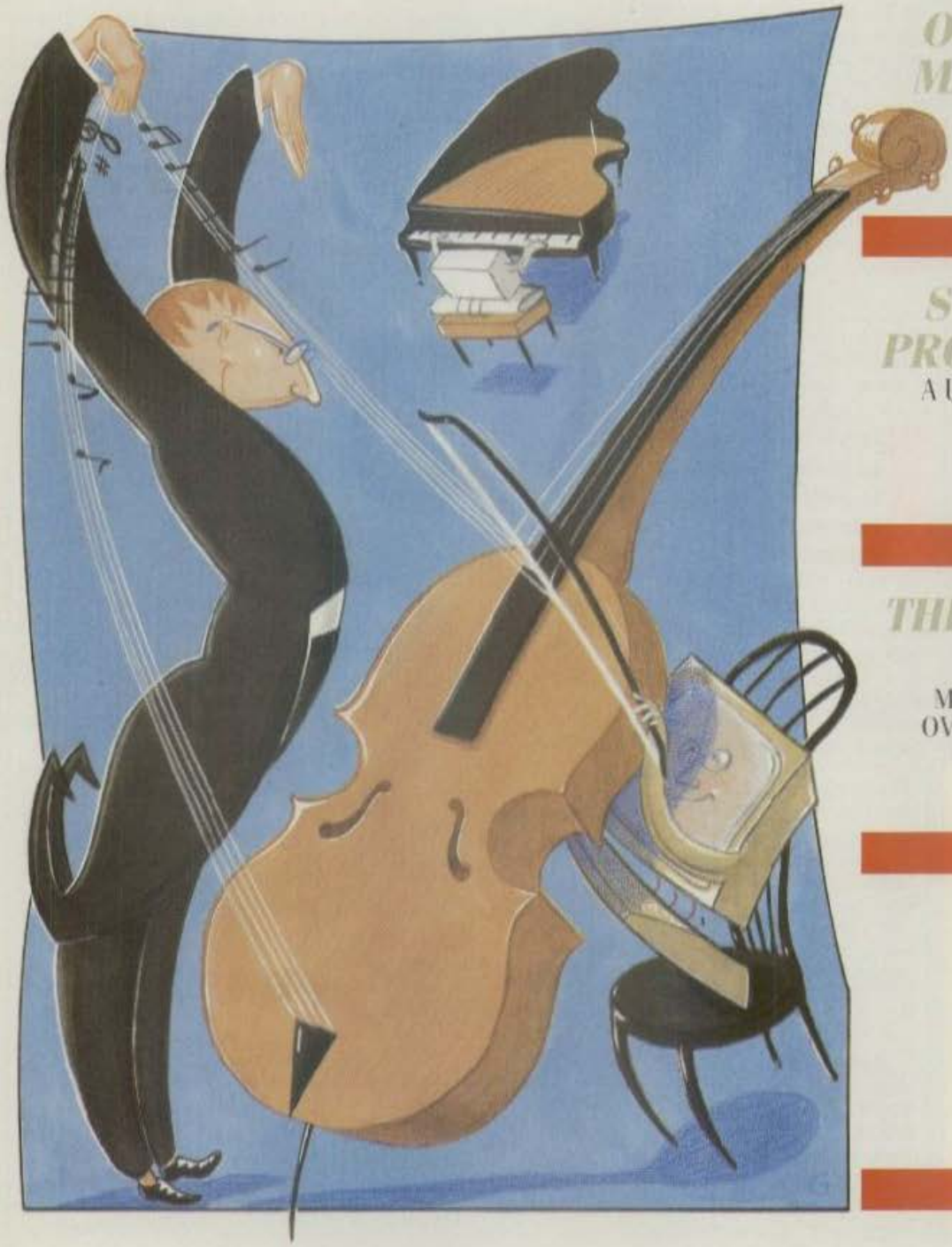
Your COMPUTER COURSE

APRIL 1986

A COMPREHENSIVE GUIDE TO PROGRAMMING – PART 2



ORCHESTRAL MANOEUVRES

A MEDLEY OF MICRO MUSIC

STRUCTURED PROGRAMMING

A USEFUL TECHNIQUE – NOT A UNIVERSAL PANACEA

THE POWER OF FORTH

MACHINE CODE, MOVE OVER – FORTH HAS THE SPEED

GUIDE TO COMPUTER JARGON

FROM E TO H IN OUR PLAIN MAN'S GUIDE

In this, the second part of the *Your Computer* Course, we develop some of the themes introduced in last month's instalment. Our main topic this month is music, in particular the use of microcomputers to control synthesisers by way of the internationally-agreed Midi standard. Combining the programming power of a microcomputer with one of the latest, low-cost music synthesisers assures the user of the best of both worlds. Even the tyro musician, with the help of some of the excellent software available, can soon be making the most melodic sounds.

In addition, this month we discuss the Forth language, putting its features into perspective, and look at the pros and cons of the structured programming debate. To some computer users, structured programming is more a religion than a technique. We take a studied view of the arguments for and against a structured approach to programming, putting the arguments of both sides in perspective.

Rounding off this month's course is the second part in John Lettice's guide to computer jargon, in which he explains terms in a way which is both frivilous and fundamental. This month he takes us through the letters E to H.

INDEX

The History of Forth 10
Glossary (E-H) 12
Midi interfacing ... 14
Structured Programming 16

Contributors:
John Lettice
Geof Wheelwright

The Forth in action

Forth has been used in a variety of applications – Geof Wheelwright describes the ma

Most of the current crop of popular computer languages available to the micro user have been around for some years. Forth is no exception. Developed in the late 1960s, it had an amazing impact when introduced to the micro world a few years ago.

For an idea of Forth it is worth looking at its history. Forth was developed by Charles H. Moore. He was working at the National Radio Astronomy Centre in Kitt Peak, Arizona, and was very frustrated with the limitations of existing programming languages.

During the next few years, Moore developed a set of programming tools capable of accepting additions. He decided to develop his tools into a control language and Forth was born. Since then, there have been numerous versions of the language, but basically the choice is between two – Forth-77 and Fig-Forth.

Moore developed his Forth system further and it is still in use today controlling the guidance system of the radio telescope at Kitt Peak. Since those early days, Forth has been used in a variety of applications. Many popular arcade games are written in Forth. Washing machine manufacturers use it to program their latest models. A variant is even used by many hospitals. Using Forth on home micros tends to be a little less exotic but it is used in a variety of ways. Control applications, such as steering a buggy, is one example but the most common use on home machines is in applications demanding a fast language, which is what Forth is – very fast.

Forth was developed by Charles Moore to control radio telescopes in Arizona.

The first point to note about Forth is that it does not use instructions or commands like Basic. Neither is the language restricted to the number of facilities with which it is purchased. Instead, Forth uses words. Every version of it has a set of those words which make up the nucleus of the language, which is referred to as the dictionary.

Just like Basic, but in a more flexible manner, a Forth program can consist of any number or combination of words. VLIST is a Forth word and its purpose is to display the contents of the dictionary. About the only other similarity between Basic and Forth is that Forth has an immediate mode where single commands – words – can be executed. For bigger programs, an editor of some kind is used to enter the source code.

The real difference is the facility of allowing the user to add to the language. For example, the following word definition will add 2 to the number on top of the stack – explained later – whenever used:

Forth has been described as a high-level, low-level language because it combines exceptional features.

```
: ADD2 2 + ;
```

The syntax of Forth is fairly straightforward. Each word starts with a colon. That is followed by a name which is used whenever the word is to be used. Next follows the definition, which is usually made up from

Forth requires numbers which are to be operated on to be on the stack prior to the operator. In other words, to perform a mathematical function it is necessary to place the operand(s) before the operator(s) and not mix them as humans do:

2 3 + is the equivalent of 2 + 3.

Humans perform calculations in in-fix notation but Forth does it in post-fix notation, known as Reverse Polish Notation. Here are some algebraic expressions in in-fix notation – normal – with their post-fix – Forth – equivalents:

Normal	Forth
A+B	AB+
A+B–C	AB+C–
(A+B)*C	AB+C*
A*B–C/D+E	AB*CD/–E+

It looks confusing at first but it is surprising how easy it is to adapt to it.

The reason why post-fix notation is used is because computers find it much simpler to work with. As a result, that also contributes to Forth speed. In Basic:

```
PRINT 1 + 2
```

requires the interpreter to scan the line, find the 1, look for an operator, find the second number, and so on.

Whenever Forth encounters an operator such as '+', '–' and so on, it ASSUMES that there ARE two numbers already on the stack. Because there is no error-checking on the stack it is for the user to make sure that there are two numbers on the stack; otherwise Forth will report with the ?STACK EMPTY error message. Reverse Polish Notation has been used for some time and some readers may remember the early Hewlett-Packard calculators which operated in that manner.

language

ıany popular programs are written in the language. advantages of the Forth alternative.

other Forth words. It is possible to create new words from machine code.

Forth has been described by many as the high-level, low-level language. That is because Forth offers such high-level control structures yet requires the user to maintain the stack which is usually a low-level operation required in machine code. The stack, and understanding how to manipulate it, is probably the single most important aspect of learning the language.

The reason is because every Forth word alters the contents of the stack in some way so it is entirely the prerogative of the user to maintain the stack, as there is no error-checking, bound-checking or such like. Even though it is a difficult concept to handle initially, stack manipulation has its rewards with programs which can run at 80 percent of the speed of machine code programs.

The stack is an area of memory used as a temporary storage space for numbers, data and such like. It operates on a first-in, last-out basis and numbers can be added to it – pushed – or removed – popped. To put the number three on the stack in Forth in immediate mode, the following is done:

```
3 <RETURN>
```

This places six on the stack. Adding more words is simple:

```
1 3 2 4 7 63 <RETURN>
```

Removing the numbers from the stack can be done in a number of ways. To print the top value on the stack the dot word '.' is used, which removes the top item of the stack and prints it to the screen:

```
. . . . . <RETURN>
```

would print

```
21
34
9
2
5
6
```

The reason for the 'backwardsness' is because the first number placed on the stack will be at the bottom and hence the last out.

The two methods of placing numbers on the stack are in the immediate mode, as just detailed, or within a colon definition – a program. Here, a word

Once learned, Forth can be used for a variety of applications ranging from fun and games to utility writing.

called ADD has been defined which adds three and five and prints the result:

```
: ADD 3 5 + . ;
```

By entering ADD, the number nine will be displayed. That is not a very good way to add two numbers so, like Basic, Forth can get the two numbers at run-time:

```
: ADD + . ;
```

To add two numbers the following is entered:

```
2 5 ADD
7 OK
```

Notice the OK which is the equivalent of the Basic Ready prompt.

Using the stack for calculations and storage will guarantee maximum efficiency when using Forth. It is possible to use variables and constants as in Basic, although at a sacrifice of a little speed.

Constants are values which do not change during the execution of the program. Defining a constant is simple:

```
6 CONSTANT SIX
```

Here a constant called SIX has been defined which contains the number six. SIX may then be used in place of the number. Variables are defined in a similar manner:

```
1000 VARIABLE LIMIT
```

This example simply sets up a variable called LIMIT, which has an initial value of 1000. Two operations need to be carried-out to place the value of the variable on to the stack. First, the name of the variable is specified which puts the address where the variables contents are stored on to the stack. The Forth word '@' (fetch) is then used which takes the address and substitutes it with the value of the variable.

LIMIT @ . would display the value of LIMIT. Altering the value of a variable requires three operations. First, the new value must be placed on the stack. That is followed by the variable name which places the address of the variable also on the stack. The '!' – store – word is then used which takes the new value and stores it at the address on the stack.

```
231 LIMIT !
```

This would alter LIMIT to 231.

There is much more to Forth than what is described here. The language has a large number of control constructs, decision-making facilities, and so on. It is a difficult language to learn but that is mainly because the syntax and method of operation is far different from Basic. Once learned, Forth can be used by the micro user for a variety of applications ranging from fun and games to utility writing.

Forth has become the second most popular language available on micros. Owners of the BBC micro have an excellent version of Forth available from Acornsoft. Melbourne House Forth for the Spectrum contains many extra graphics and sound facilities, while Commodore 64 owners have at least five Forth packages from which to choose. I can recommend the Fig-Forth cartridge from Audiogenic. Finally, Amstrad users cannot go wrong with Abersoft Forth, which is one of the fastest Forths I have seen on a Z-80 system.

The following Basic/Forth comparison demonstrates the speed of Forth. The two programs are identical in operation and perform the following:

* Clear the text screen
* Fill the colour map with the value 1, i.e., white.
* POKE to the screen display 26 complete screenfulls.

The result is that a screenful of As will be displayed, then Bs and so on. In Basic the test took 115 seconds to complete on the Commodore 64 and in Melbourne Forth it took 10.23 seconds:

Basic

```
10  PRINT "[CLS]"
20  FOR C=55296 TO
    56296
30  POKE C,1
40  NEXT C
50  FOR L=1 TO 26
60  FOR I=1024 TO 2039
70  POKE I,L
80  NEXT I
90  NEXT L
```

Forth

```
10) : BM1 CLS
20)     56297 55296 DO
30)       1 I C!
40)     LOOP
50)     27 1 DO
60)       2040 1024 DO
70)         J I C!
80)       LOOP
90)     LOOP ;
```

Line 10 in both programs clears the screen. Line 20 sets up an index; the values of the index are the start and end addresses of colour RAM on the 64. Notice in the Forth example that the end address PLUS ONE is specified. Line 30 in both examples POKEs values into the variable C. Similarly, the second loop POKEs values into L and I.

Your COMPUTE

Our off-beat guide to computer-speak from John Lettice's individualisti

■ **Edit:** To insert, move, correct or delete vital parts of a program accidentally.

■ **Editor:** A program or part of a program designed specifically to allow you to delete things accidentally faster.

■ **Electronic publishing:** See Rupert Murdoch. Seriously, though, the advent of laser printer, together with high-resolution graphic screens, means it it feasible to produce finished magazine art work – see *Your Computer* – with only a laser printer and a micro.

■ **Element:** The important part of an electric kettle; alternatively, a single piece of data.

■ **ELSE:** An extension to the IF . . . THEN loop construction which allows for a "false path." So, if I have enough money, THEN I will buy a 520ST. ELSE I will have to settle for an Electron.

■ **EMail:** Short for electronic mail, the transfer of written information from one terminal to another, usually by means of telephone lines and modems. It is a fast and hassle-free means of communication. The copy for this article, for example, was delivered by electronic mail through an Olivetti M-21.

■ **Embedded code:** A piece of machine code inserted to make an otherwise unintelligible program completely unintelligible. Ostensibly it is there to do a job which a Basic listing cannot do but we all know the real reason.

■ **Emulate:** 1, Obsolete term meaning to wish to be Guy Kewney – see *Cambridge Atlas of Ancient Microcomputing*. 2, To imitate a computer to the extent that the emulator can run the same software, data and so on. For electronic mail purposes, a computer will often run terminal emulation software. The term emulator is also used loosely to refer to computers sold on the basis that they work exactly like other computers, e.g., IBM emulators, lookalikes or clones.

■ **Encode:** To put into code which is understood directly by the computer, or to render a program unintelligible.

■ **Encrypt:** Not exactly the same as encode, as it refers to data being made deliberately unintelligible to humans for security reasons. If we translate so that micros can understand things, and micros translate so that we cannot, it is fairly clear who is boss.

■ **END:** The last entry in a pedant's program, telling the computer there is nothing to follow. They generally stop anyway.

■ **Eniac:** One of the first true computers, developed in the U.S. during World War II. It filled a whole room and its computing power was similar to that contained in a Habitat dimmer switch. Well, it was a little more than that then.

■ **EOF:** Technical term used by rugby prop forwards during scrum, along with EEOOF and GRRRK. In computing it stands for End Of File, a character recorded on magnetic media – disc or tape – to mark the end of a particular file.

■ **EPROM:** Erasable Programmable Read Only Memory. It is a chip which allows you to "burn" a program or operating system – a computer's operating system, or O/S is essentially a suite of programs – on to it. The advantage of this kind of chip is that it is easy to change and is much loved by manufacturers who have not finished a machine's O/S but want to sell it anyway.

■ **Error-checking software:** Software which checks the validity of data for you, ranging from spelling checkers upwards.

■ **Error message:** In the bad old days it was a number which could be looked up in the manual and would thus tell you what was wrong with a program. The more user-friendly models now put the words on the screen but that does not make them easier to understand.

■ **Escape:** Many computers have an Escape key which permits you to break out of a program which is going wrong. It doesn't work? Try Shift/Escape. Well, try Control/Shift/Break/Escape. All the best programmers are reduced to pressing as many keys as possible prior to giving up and switching the thing off.

■ **Escape character:** Disguise used by British officers to get away from Colditz, e.g., "Ich weiss kein Deutsch, I yam Swedish engineer." Also a character used with one or more succeeding characters which will make the computer understand succeeding characters differently. Escape characters, when sent to printers, can change the type or format of the output.

■ **Ethernet:** A data communications network developed by a number of U.S. companies with the pious and laughable hope that it would be accepted as a computer industry standard.

■ **Execute:** Micros do not do jobs, they execute them. Execute, or EXEC, is used generally with reference to the direct memory access which calls a machine code routine. CALL is also used, as is RANDOMIZE USR on Sinclair machines, but murder, top and assassinate are uncommon.

■ **Expansion card:** A circuit board produced by manufacturers of under-powered computers to allow them to sell you more equipment.

■ **External interrupt:** Pounding on the door heard while playing Super Zaxxon. Also an interrupt produced by an external hardware device. See **Interrupt.**

■ **Fanfold:** Origami term used in CAD systems. Also continuous feed paper, which you can fan and fold – or put in the printer.

■ **Fibonacci series:** A series of integers where each is the sum of the two preceding it in the series, e.g., 1,2,3,5,8,13. Mathematicians get excited about them and write articles with many pretty diagrams based on Fibonacci.

■ **File:** A collection of data, generally stored on an external device, e.g., tape file, disc file, nail file and so on.

■ **File attribute:** A feature of a file which can be changed by the user, often making it unreadable. File characteristics cannot be changed.

■ **File maintenance:** Largely theoretical term governing the process of keeping files up-to-date.

■ **File specification:** A way of identifying a file which generally will include name, type and location on disc. Abbreviated to filespec.

■ **Firmware:** Company tie.

R COURSE

**ontinues as we present another extract
c explanation of computing terms.**

Also programs stored on a chip which are not lost when the machine is switched off, as software is.

■ **Flag:** A marker used to denote a particular thing about a computer's state. Red flag on the Pentagon indicates the Russians are there, Stars and Stripes indicates they are not.

■ **Floppy disc:** You should not flop even the ones that do flop. They are flexible discs which store data in magnetic form, although the newer types are in rigid casing.

■ **Flowchart:** A diagram used to represent the flow of a program, or more likely a projected program, sometimes used to avoid having to write the thing in the first place. If you have seen a flowchart with a box labelled "really amazing graphics sequence", you will understand this.

■ **Font:** A font defines the size and design of a set of print characters and at weekends is used by the vicar for christenings. That is why print unions have chapels rather than branches.

■ **Format:** A command used to order a disc so that it can be used by a particular computer. As it wipes all the data from the disc it is used most commonly on the most vital disc you have rather than the blank one which you meant to use.

■ **Forth:** Every so often someone goes bankrupt trying to sell a micro which runs the Forth language rather than Basic, because Forth is faster, better-structured. An language which uses something called "Reverse Polish Notation" can't be all bad, but it will never catch on.

■ **Fortran:** Language used in science and engineering by people who don't eat quiche.

■ **Front end:** A user-friendly front end is supposed to be a system which is easy to learn and understand and which allows you to operate the system without meeting the operating system. It can also function as a device to prevent you learning about the machine, or to con you into buying a turkey because it looks easy.

■ **Full duplex:** A mode of transmission which allows data to move in both directions simultaneously, like patting your head and rubbing your stomach at the same time.

■ **Garbage:** Used generally of the output of a computer when it should make sense but it does not. It is more common in the programming input but programmers don't see things that way.

■ **Gigabyte:** One billion bytes. Techies salivate over the things they could do with this kind of memory, although they tend to forget how long a lifespan they would need.

■ **GIGO:** Garbage In, Garbage Out. One of the two basic truths of computing, the other being Nobody Ever Got Fired For Buying IBM, although NEGFFBI does not really roll off the tongue.

■ **Glitch:** Technically a sudden change in voltage level but used indiscriminately of anything which goes wrong, e.g., "Waiter, there's a glitch in my soup."

■ **Global variable:** A variable which is defined for the whole of a program, as opposed to a local variable, which applies only to a part.

■ **GOTO:** Command which jumps the flow of a program to another part of it. Structured programmers deplore its use, preferring to foul their programs in other more structured ways.

■ **Half duplex:** A method of data transmission where data can be sent only in one direction at any given time.

■ **Hand assemble:** To assemble a program without having an assembler do it for you.

■ **Handshaking:** a signal exchange between a computer and an I/O unit used to synchronise the operation of the two. Computers, being more polite than you would think, tend to shake hands before they get down to business.

■ **Hands on:** The experience whereby you touch, learn about, drop gob-stoppers down and generally break the real thing, as opposed to reading about it.

■ **Hard copy:** Extremely deep, unintelligible program listing. Also a paper printout of information.

■ **Hard disc:** Concurrent DOS really but most people think of a hard disc as a rigid disc fixed in a sealed environment. The disc rotates faster than a floppy and has greater storage capacity. Also known as a **Winchester.**

■ **Hardware:** Computer hardware is the physical part of the system, as opposed to software, the thinking bit. For some reason computer hardware is not so tough as proper hardware, i.e., saws, hammers and so on.

■ **Hardware interrupt:** An interrupt requested by a peripheral device.

■ **Head:** The reading head of a storage unit, either tape or disc.

■ **Head crash:** An analogous to dropping the needle on to a record from a great height. The head on a Winchester is cruising round, minding its own business, when suddenly it bumps into this monstrous piece of dust and . . .

■ **Header:** Part of a tape or disc which contains details of the files stored there.

■ **Hertz:** Unit of frequency, abbreviated to Hz, governing, among other things, the speed or clock speed of a micro.

■ **Hex:** Short for hexadecimal, the counting system based on 16 rather than 10. It is used as a shorthand way of writing binary.

■ **High-level language:** Computers word faster if you talk their language but you can't. A high-level language is a programming language like Basic which allows you to do complex things while having them interpreted for the computer.

■ **Hook:** Usually used of software, hook describes an expansion capability which will allow programmers to add facilities later. Alternatively, a capability in the software which will permit the manufacturer to put back the piece it ripped out at the last minute as soon as it can get it to work.

■ **Host computer:** A large computer which has a smaller computer in communication with it.

■ **Housekeeping:** A series of routines a micro performs at set times in its operation, typically checking its status before becoming available for operation. Computers wash the dishes before they go to work.

Orchestral Manoeuvres

The Musical Instrument Digital Interface standard for connection between electronic musical instruments and computers has been in existence only a few years but it is appearing on machines developed by everyone from Atari to Sinclair, with a major participation from many of the Japanese companies which make the musical keyboards.

It is a serial, digital computer interface and allows up to 16 instruments to be chained for control by the computer. The fact that it is a serial interface has meant that when Sinclair decided to take Midi to the new Spectrum 128 it could double-up the Midi interface and the serial interface on the same socket – it will require only slightly different cables to use Midi instead of the serial port.

There are popular misconceptions about Midi and what it is supposed to do. It does not let your micro 'play music' but instead permits the machine to act as a controller of instruments – a kind of electronic conductor.

At the launch of the Spectrum 128, for example, a large table was set up purporting to show what the Spectrum and Midi could do together. It contained several large music keyboards, loudspeakers and a few amplifiers; almost lost in the middle of it was a lonely Spectrum 128 connected to a TV set with musical notes flashing on the screen.

The major point is that your Midi operation will be only as good as the instruments you plug into your Midi port – the existence of the port does not enhance the musical ability of your machine significantly – and the software you use. All the Midi interface essentially allows you to do is to control the flow and type of control codes sent to the instruments which regulate things such as echo, delay and reverberation.

It sounds obvious but you also need to make sure that the instruments you want to use with your Midi micro also contain the Midi interface. There have been previous attempts to construct music/computer interfaces but only a Midi-standard interface on your instruments will work with your Midi micro.

Cost will also be an important factor in how your Midi system is set up. There is no point in spending money for a micro with a built-in Midi – or buying one of the many new Midi interface add-ons for machines like the BBC micro – if you do not have the funds for some worthwhile Midi instruments to plug into it. Midi equipment does not mean an average £50 add-on; it usually runs into several hundred pounds.

Good keyboards are vital for most music input when using Midi. You should look for something which is flexible, offering a large number of instruments, and a full-size keyboard. If you are trained in using a full-size piano keyboard, you may find that many Midi keyboards are too small for you to use.

With many Midi systems you will also need a good amplifier and speakers into which to plug the Midi micro and keyboard. Be very careful to ensure that the speakers into which you plug your Midi into are powerful enough to handle it. Too much power put through cheap speakers or a bad amp could ruin them.

The future for Midi looks bright, with large numbers of manufacturers introducing new equipment which conforms to the Midi hardware standard and a good deal of new Midi software being developed. There is even talk of providing Midi interfaces for lighting and graphics devices so that light shows can be attached to Midi set-ups.

UMI-2B interface

Pop music has long been associated with complex electronics and expensive musical equipment, yet the success of groups like Ultravox, Tears For Fears and, most recently the Norwegian band A-Ha, may lie in something as inexpensive as the BBC micro.

The secret of the system lies in the development of Music Instrument Digital Interface and an interface called the UMI-2B. Midi has become a standard in the music industry and can be found at the back of many modern electronic keyboards, drum synthesisers and multi-track tape recorders.

Acting as a serial digital computer interface, Midi allows up to 16 instruments to be chained together and controlled by each other or by a computer. While a basic Fairlight – the Fender guitar of the modern electronic pop world – makes extensive use of Midi, the price of it has risen from £28,000 to more than £50,000. Meanwhile, the UMI-2B, designed for exclusive use with the BBC micro, costs only £499.

Complete with software on ROM, a rather scanty manual and all the necessary connections, the UMI is housed in a BBC beige box which plugs into the 1MHz bus and the user port. With the BBC switched on a simple *UMI command takes you instantly into the recording studio.

The easiest way to describe the system is to run through the creation of a song. My musical career ceased abruptly in my juvenile piano lesson days, since which time my greatest rendition was using a comb and tissue paper. Yet the UMI system, combined with the simplest of the Midi keyboards – the Casio CZ101 – let me develop a complete re-mix of the Axel-F theme song from the film Beverly Hills Cop.

Having played with the keyboard and any drum machine to hand, the aspiring musician should have an idea of how a song will sound. I already knew how the song should sound because I possessed the record.

Having selected real-time input from the main menu, you have to define the length of your first sequence of notes. That can be almost any length from one to 16 bars, and obviously can include chords. Pressing the button on top of the UMI gives a four-beat count-in before you start playing. Otherwise recording begins the moment you press the keys. Alternatively, if your playing ability is restricted, step time allows each pattern to be put together, note by note.

Remember that each note played is stored by the BBC as a number and the BBC makes no sound. The tunes played by the computer will sound different depending on which keyboards are connected and which instruments are selected on those keyboards.

Having defined the first section, it can be played back while you play an accompaniment. That, in turn, will be recorded as a separate section. Having produced up to 127 of those patterns, you can use them to create the song.

The 16 channels act as 16 sequences of music which can all be played together. That is more than most people can cope with but simply stringing together sequences in two or three channels – one as the base line and another as the main tune – you can produce complete songs. The UMI-2B has put musical composition within reach of most aspiring bands, while remaining powerful enough to attract bands like Blancmange, a Flock of Seagulls, and even Queen. One particular fan of the system is Vince Clarke, formerly of Depeche Mode, who has even used the system live on stage when touring the country.

New bands can produce complete songs using only a couple of keyboards.

The system also saves considerable expense normally incurred when hiring a recording studio. New bands can produce complete songs using one or two keyboards at home, before taking the floppy disc to the recording studio and slotting it into the disc drive. It is much more attractive to a record company than yet another cassette tape.

Thanks to The Rock Shop in Camden, London, for loaning us the UMI and keyboard for the course.

A complete UMI-2B system. ▼

To structure or not – that is the question

Geof Wheelwright looks at the great programming debate

One of the great debates among programmers centres on the need for structured programming, which divides the large number of jobs in any program into small, manageable portions. The logic behind that is obvious. Like anything you build from a series of parts – whether it is a car, a television or a computer program – the ability to strip it into its essential components makes it easier to service. Your car, for example, is designed so that if the water pump breaks down, you have only to replace that and not the entire engine. The key, of course, is ensuring that you can diagnose the fault as being caused by the water pump.

With computer programming, you construct all the constituent parts and thus have a chance to test each of them individually before you fit them together. Most computers also have the added advantage of giving error messages when your program malfunctions after you have fitted together the constituent parts. That tells you immediately where to find the error and which part of the program needs repairs.

All too often we see clever little pieces of software brought together in a huge stadium of code, only to riot against one another and cause on screen havoc. Structured programming can result in code overkill.

If you have read your manual and done some programming, you will probably know all this. What you may not be familiar with are the pros and cons behind the structured programming debate. We present both sides of that argument, with the opposition opening the debate on the motion "PROCedures, GOSUBs and a structured approach is the ONLY way to program".

CON: While there is a good deal of merit to the idea that modular programming, in which the main objective of the programmer is to string together many little pieces of program to make one big program, eases debugging and design, it can often make it more difficult. Without careful attention to the original purpose of the program, programmers can become immersed in the detail of various routines and waste time looking for more and more elegant ways to solve relatively inconsequential problems in the routines which drive the main program.

That can happen sometimes to such an extent that the whole program can become subservient to the grace and elegance of one of the programmer's pet routines. A realistic design specification and a plan of attack which allows only a certain amount of time for the design of each routine would make better and more prolific programmers of all of us.

All too often we see clever little pieces of software brought together in a huge stadium of code, only to riot against one another and cause on-screen havoc. Like a writer who tries to string together too many bad metaphors and forgets the original topic, each routine is not on its own – it must work well with the rest to produce a good result.

Structured programming can also result in code over-kill. When the program is divided into so many tiny little jobs, tasks can be duplicated, new variables being created where old ones would suffice, with the general speed of the program being badly-affected because the machine is spending too much time doing unnecessary tasks.

Structure and planning are important but do not become so obsessed with producing all the little routines which make up the program that you forget about the need to make them all work together in the end.

PRO: If you are to do anything in life properly you need a plan. That is probably more true of computer programming than almost any other pursuit you might consider. With no overall blueprint, you constantly have to fix one portion of programming as it falls out of kilter with the next.

A good structured programmer makes copious use of PROCedures, where they are available on machines such as the BBC micro and the Sinclair QL; or GOSUBS – where they are not, such as on the Commodore 64 and the Sinclair Spectrum. By designating each of the major jobs in the program as a procedure or subroutine, each can be tested and inspected individually. Some may suggest that leads people to spending far too much time with each but I would submit that it brings out the best possible in each section of the software.

If you spend a long time on one exceptional method of doing something – such as the 3-D graphics effects on Spectrum *Ant Attack* or the software-driven speech synthesis in Commodore 64 *Ghostbusters* – that technique may prove to be more important than the rest of the program. A structured program is a happy program.

It is, of course, your prerogative but, overall, the proponents of the structured approach tend to win the debate every time. Without being too santimonious about it, however, you should bear in mind that all those lovely pieces of structured programming will have to work properly together in the end and no amount of routine-writing will change that.

If you are to do anything properly you need a plan. A good structured programmer makes copious use of PROCedures or GOSUBs.